IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

EV 316936132

VALIDATION OF XML DATA FILES

Inventors:

Jean-Marie H. Larcheveque, Arungundram Narendran, Prakash Sikchi, Alexei Levenkov, Adriana Ardeleanu, Andrey Shur, Alessandro Catorcini, Nora S. Selim and Kamaljit S. Bath

ATTORNEY'S DOCKET NO. MS1-1698US

CROSS REFERENCE TO RELATED PATENT APPLICATION

This is a continuation-in-part of a co-pending United States Patent Application having serial no. 10/402,640, a filing date of March 28th, 2003, and an attorney's docket number of MS1-1341US, for SYSTEM AND METHOD FOR REAL-TIME VALIDATION OF STRUCTURED DATA FILES of Jean-Marie H. Larcheveque et al. This co-pending United States Patent Application is commonly assigned herewith and is hereby incorporated herein by reference for all that it discloses.

TECHNICAL FIELD

This disclosure relates to using XML elements for validation of structured data files.

BACKGROUND

Extensible markup language (XML) is increasingly becoming the preferred format for transferring data. XML is a tag-based hierarchical language that is extremely rich in terms of the data that it can be used to represent. For example, XML can be used to represent data spanning the spectrum from semi-structured data (such as one would find in a word processing document) to generally structured data (such as that which is contained in a table). XML is well-suited for many types of communication including business-to-business and client-to-server communication. For more on XML, XSLT (eXtensible Style-sheet Language Transformation), and XSD (schemas), the reader is referred to the following documents which are the work of, and available from the W3C (World Wide Web consortium): XML Schema Part 2: Datatypes; Extensible Markup Language

(XML) 1.0 second edition specification; XML Schema Part 1: Structures; and XSL Transformations (XSLT) Version 1.0.

Before data can be transferred, however, it must first be collected. Electronic forms are commonly used to collect data. Electronic forms collect data through data-entry fields, each of which typically allows a user to enter data. Once the data is received, it can be stored in an XML data file. The data from a particular data-entry field typically is stored in a particular node of the XML data file.

Users often enter invalid data into data-entry fields, however. Invalid data, when stored in a data file, can misinform people and cause unexpected behavior in software relying on the data file. Because of this, businesses and individuals expend extensive time and effort to prevent invalid data from making its way into XML data files.

One such way to help prevent invalid data from corrupting an XML data file is to validate the data before the data file is saved or submitted. By validating the data file before it is saved or submitted, invalid data can be corrected before it is permanently stored in the data file or used by another application. Validation typically is performed when a user attempts to submit or save the entire form, and is thus performed on a group of individual data fields at one time.

One of the problems with this manner of validating data is that the user receives a list of errors disjointed from the data-entry fields from which the errors arise. These errors may be difficult to relate back to the data-entry fields in the electronic form, requiring users to hunt through the data-entry fields to find which error from the list relates to which data-entry field in the electronic form.

Another problem with this manner is that even after the user determines which error from the list relates to which data-entry field, the user may have to expend a lot of effort to fix the error if the error notification is received well after the user has moved on. Assume, for example, that the user has entered data from a 400-page source document into ninety-three data-entry fields. Assume also that once finished, the user attempts to save or submit the electronic form. A validation application then notifies the user of sixteen errors. After finding that the first error relates to the eleventh data-entry field out of ninety-three, the user will have to go back through the 400-page document to find the data that he or she was supposed to correctly enter into the eleventh data-entry field. This manner of validation can require extensive hunting through large or numerous source documents to fix old errors, wasting users' time.

Even worse, the validation application may return only the first of many errors. For this type of validation application, a user has to go back and fix the first error and then re-save or re-submit. If there are many errors in the electronic form—as is often the case—the user must go back and fix each one separately before re-saving or re-submitting to find the next error. If there are even a few errors, this process can take a lot of time.

Another problem with this process is that if the user submits the electronic form to a server, it taxes the server. A server can be slowed down by having to validate electronic forms, reducing a server's ability to perform other important tasks.

In addition to these problems, the current way of validating data for structured data files can allow some data that is not desired. While this allowance

of undesired data can sometimes be prevented, doing so can require extensive time and sophisticated programming abilities.

For these reasons, validation of data for XML data files can require a lot of a data-entry user's time and tax servers. In addition, without a skilled programmer expending considerable effort, significant amounts of undesired data can get through.

SUMMARY

In the following description and figures, XML elements and accompanying processes are disclosed. These elements and their accompanying processes are used to specify validation constraints and associate these constraints with nodes of an XML data file. These validation constraints are used by a real-time validation tool to validate and invalidate nodes of an XML data file.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 illustrates a system with a display screen, computer, and user-input devices. The system implements a method for validating data for structured data files.

Fig. 2 illustrates an exemplary screen display showing a blank electronic form having data-entry fields.

Fig. 3 is a flow diagram of an exemplary process for real-time validation of data for a structured data file.

Fig. 4 illustrates an exemplary screen display showing an electronic form with a filled-in data-entry field.

Fig. 5 illustrates an exemplary screen display showing an electronic form with a data-entry field having an invalid entry.

Fig. 6 illustrates an exemplary screen display showing an electronic form with a data-entry field having a modeless invalid entry and a dialog box.

Fig. 7 illustrates an exemplary screen display showing an electronic form with a data-entry field having a modal invalid entry and a dialog box.

Fig. 8 illustrates an exemplary screen display showing an electronic form having many filled-in data-entry fields, one of which contains a modal invalid entry, and a dialog box.

Fig. 9 is a flow diagram of an exemplary process for using XML elements to validate data in an XML data file.

Fig. 10 illustrates an exemplary error-condition XML element.

Fig. 11 is a flow diagram of an exemplary process for associating a node with an error message using an XML element.

Fig. 12 illustrates an exemplary error-message XML element.

Fig. 13 is a flow diagram of an exemplary process for use of an XML element to provide an error message for a node subject to a schema-based validation rule.

Fig. 14 is a flow diagram of an exemplary process for use of an XML element as an aid in executing code when a node of an XML data file is modified.

Fig. 15 illustrates an exemplary event-handler XML element.

Fig. 16 is a flow diagram of an exemplary process for adding validation rules for use by a real-time validation tool.

Fig. 17 illustrates an exemplary custom validation screen, including a development copy of an electronic form and a properties box.

Fig. 18 illustrates an exemplary custom validation screen, including a development copy of an electronic form, a properties box, and a validation option box.

Fig. 19 illustrates an exemplary custom validation screen, including a development copy of an electronic form, a properties box, a validation option box, and a preset validation selection box.

Fig. 20 illustrates an exemplary custom validation screen, including a development copy of an electronic form, a properties box, a validation option box, and a preset validation selection box.

Fig. 21 illustrates an exemplary custom validation screen, including a development copy of an electronic form, a properties box, a validation option box, a preset validation selection box, and a node selection box.

Fig. 22 illustrates an exemplary custom validation screen, including a development copy of an electronic form, a properties box, a validation option box, and a preset validation selection box.

Fig. 23 illustrates an exemplary custom validation screen, including a development copy of an electronic form, a properties box, a validation option box, and a preset validation selection box.

Fig. 24 illustrates an exemplary custom validation screen, including a development copy of an electronic form, a properties box, and a validation option box.

Fig. 25 illustrates an exemplary script entry screen for entry of a script-based validation rule.

Fig. 26 is a block diagram of a computer system that is capable of supporting use of XML elements in validating XML data files.

The same numbers are used throughout the disclosure and figures to reference like components and features.

DETAILED DESCRIPTION

The following disclosure describes an easy and intuitive way for a user to correctly edit structured data files by notifying the user of her errors as she makes them. As a user enters data into a data-entry field of an electronic form, a real-time validation tool validates the data to ensure that the data is valid. If the data is valid, the user can continue on to the next data-entry field. If the data is not valid, the real-time validation tool may allow the user to continue on or may not, depending on the error. The real-time validation tool, whether it allows the user to continue or not, does not allow the user to output the data into a structured data file until it is valid.

In cases where the structured data file is an XML data file, the real-time validation tool can validate data entered into nodes of the XML data file using XML elements, attributes, and XPath expressions. These XML elements, attributes, and expressions can be used by the real-time validation tool to determine whether or not a node is valid and what error message to display and in what circumstances. Some of these can also point to executable code that is used to perform operations when a node is modified.

The following disclosure also includes a description of a way for a developer to create custom validation rules for use by the real-time validation tool. The real-time validation tool uses validation rules to determine when data entered is valid or invalid. A developer can adjust or add new rules for use by the real-time validation tool. A developer can, for instance, add a validation rule, set

whether a user can continue or not if the rule is violated, decide what information is provided by the real-time validation tool to the user when the rule is violated, and determine how these are done.

Creating custom validation rules will be discussed in greater detail in the later parts of the detailed description.

For discussion purposes, the real-time validation tool is described in the context of a single computer, user-input devices, and a display screen. The computer, display screen, and user-input devices will be described first, followed by a discussion of the techniques in which these and other devices can be used.

Exemplary Architecture

Fig. 1 shows an exemplary system 100 used to facilitate real-time validation of structured data files. This system 100 includes a display 102 having a screen 104, user-input devices 106, and a computer 108.

The user-input devices 106 can include any device allowing a computer to receive a developer's input, such as a keyboard 110, other device(s) 112, and a mouse 114. The other device(s) 112 can include a touch screen, a voice-activated input device, a track ball, and any other device that allows the system 100 to receive input from a developer.

The computer 108 includes components shown in block 115, such as a processing unit 116 and memory 118. The memory 118 includes computer-readable media that can be accessed by the computer 108, and can be volatile and nonvolatile, removable and non-removable, or implemented in any method or technology for storage of information. The memory 118 includes applications shown in Fig. 1, such as an operating system 120 and a real-time validation tool

122, which includes a user interface 124. The computer 108 communicates with a user and/or a developer through the screen 104 and the user-input devices 106.

The real-time validation tool 122 facilitates real-time validation of data for structured data files and is executed by the processing unit 116. The real-time validation tool 122 is capable of validating data entered into an electronic form as it is being entered by a user. Thus, with each new piece of data entered, the real-time validation tool 122 can check whether or not that data is valid and respond accordingly.

In one implementation, the real-time validation tool 122 is capable of validating numerous pieces of data as a group, rather than one-at-a-time as each is entered by a user. Thus, after numerous pieces of data have been entered, such as when a user attempts to submit an electronic form after entering data into many data-entry fields of the electronic form, the real-time validation tool 122 validates each piece of data that needs to be validated.

The real-time validation tool 122 can respond to an invalid entry by informing the user of invalid data entered and allowing or not allowing the user to continue editing the electronic form. (The real-time validation tool 122, however, can also validate loaded data as well). If the real-time validation tool 122 stops the user from continuing on to the next data-entry field, it alerts the user of the error. To make the error easy to fix, the real-time validation tool 122 can inform the user about the error, such as why the data entered is incorrect or what type of data is correct. The real-time validation tool 122 can alert the user through an alert containing information, such as a dialog box in a pop-up window. It can also alert the user with graphics, such as a colored box encasing the invalid data-entry field, or in other ways, like rolling back the data in that data-entry field or keeping the

user's cursor in the data-entry field. These and other ways of notifying the user and controlling the user's actions are designed to make the user's entry and correction of data as easy and intuitive as possible.

The real-time validation tool 122 can also allow the user to continue after entering invalid data. In many cases, stopping the user from continuing on to the next data-entry field is counterproductive. The user may not yet have the correct data, or may find it easier to come back to fix all his errors at once, rather than as-he-goes. The real-time validation tool 122 does, however, notify the user that the data entered into that data-entry field is invalid. By so doing, the real-time validation tool 122 informs the user but allows the user to decide if he or she wishes to fix the error now or later. The real-time validation tool 122 can notify the user in various ways, such as those discussed above, as well as particularly un-intrusive ways, like surrounding the data with a red, dashed-line border.

Electronic Forms, Solutions, and Structured Data Files

A view of a structured data file is depicted on the screen 104 through execution of the structured data file's solution. The data file's solution is one or more files (e.g., applications) used to enable a user to edit the structured data file, and may include logic and viewing applications. To edit the data file in a user-friendly way, the data file's solution contains a viewing application, such as an electronic form. This viewing application gives the user a graphical, visual representation of data-entry fields showing previously entered data and/or blank data-entry fields into which the user can enter data. A data file typically has one solution (though solutions often contain multiple files), but each solution often governs multiple data files.

Fig. 2 shows a display screen 200 including a blank electronic form 201 entitled "Expense Report", which is generated by a solution. This expense report 201 contains data-entry fields in which a user can enter data. These data-entry fields map to a structured data file, so that the data entered into the form can be held in the data file (and eventually stored, once confirmed valid). They can be stored one-by-one after each is confirmed valid, in total after all are confirmed valid and the electronic form is submitted or saved, or in groups after each entry in the group is confirmed valid. Data not yet stored in a data file can be held in various locations and ways, temporarily—such as in the data file (without it being saved), or permanently—such as in an auxiliary file.

This solution presents the expense report 201 electronic form but also contains logic that governs various aspects of the expense report 201 and the data file. In a report date data-entry field 202, for instance, the solution presents the data-entry field as a white box within a gray box, provides a description of the data desired with the text "Report Date", and contains logic requiring that the user enter only numbers. This logic, which can be or use a schema governing the structured data file, can be used by the real-time validation tool 122 in validation rules used to validate data. The solution may also contain other files used by the real-time validation tool 122 for validating data, such as files containing XML elements for validating nodes of an XML data file.

Validation rules are employed by the real-time validation tool 122 to ensure that the right kind of data is being entered before it is stored in the structured data file. A user's business manager attempting to analyze expenses with an expense code, for instance, would like the structured data file to have numbers entered into an expense code data-entry field 204. The manager may not be able to determine

how an expense should be analyzed if the expense code entered is invalid because it contains letters.

Each solution can be one file or contain many files, such as a presentation file or files used by the real-time validation tool 122 for validation rules. Files used for validation will be discussed in greater detail below. The presentation file is used to present or give a view of an electronic form enabling entry of data into a structured data file, such as a visual representation of the structured data file (blank, in this case) by the expense report 201 electronic form. In some implementations, the presentation file is an XSLT or CSS (Cascading Style Sheet) file, which, when applied to a structured data file written in XML, generates an XHTML (eXtensible Hyper-Text Markup Language) or HTML (Hyper-Text Markup Language) file. XHTML and HTML files can be used to show a view on the screen 104, such as the expense report 201 of Fig. 2.

Like solutions, structured data files can come in various types and styles. As mentioned above, structured data files can be written in XML or some other language. Structured data files, however, are structured—the data they contain is oriented relative to each other. Structured data files can be modular and/or hierarchical (such as a tree structure), for instance. In a hierarchical structure, nodes of the structured data file are designed to contain data or govern other nodes designed to contain data. Nodes designed to contain data can be mapped to particular data-entry fields, so that the data entered into a data-entry field is slotted for the appropriate node. Because nodes and data-entry fields are mapped to each other, the real-time validation tool 122 can determine what node a developer or user is attempting to select or edit when a data-entry field, rather than the node itself, is selected.

The above devices and applications are merely representative; other known devices and applications may be substituted for or added to those shown in Fig. 1. One example of another known device that can be substituted for those shown in Fig. 1 is the device shown in Fig. 26, which will be discussed later.

Validating Data From A User, Including In Real-Time

Overview

A system, such as the system 100 of Fig. 1, displays an electronic form with data-entry fields to allow a user to enter data. The user can enter data in a data-entry field and know, as he does so, whether or not the data entered is valid or invalid. By so doing, the system 100 provides an easy, intuitive, and efficient way for a user to enter and correct data intended for a structured data file.

Fig. 3 shows a process 300 for validating data entered into an electronic form. The process 300 is illustrated as a series of blocks representing individual operations or acts performed by the system 100. The process 300 may be implemented in any suitable hardware, software, firmware, or combination thereof. In the case of software and firmware, the process 300 represents a set of operations implemented as computer-executable instructions stored in memory and executable by one or more processors.

Notifying A User of Errors in Real-Time

At block 302, the system 100 displays an electronic form having data-entry fields. The electronic form can be blank or contain filled data-entry fields. The expense report 201 electronic form in Fig. 2 is an example of a blank electronic form.

The system 100 displays an electronic form in a manner aimed at making a user comfortable with editing the electronic form. It can do so by presenting the electronic form with user-friendly features like those used in popular word-processing programs, such as Microsoft® Word®. Certain features, like undoing previous entries on command, advancing from one data-entry field to another by clicking on the data-entry field or tabbing from the prior data-entry field, cut-and-paste abilities, and similar features are included to enhance a user's data-entry experience. For example, the system 100 displays a blank electronic form having some of these features in Fig. 2, the expense report 201 electronic form.

At block 304, with the electronic form presented to the user, the system 100 enables the user to input data into a data-entry field. The user can type in data, cut-and-paste it from another source, and otherwise enter data into the fields. The user can use the user-input devices 106, including the keyboard 110, the other device(s) 112 (such as a touch screen, track ball, voice-activation, and the like) and the mouse 114.

In Fig. 4, for example, the user enters "1/27/2002" into the report date data-entry field 202 of the expense report 201.

At block 306, the system 100 receives the data entered into the data-entry field by the user. The system 100 receives the data from the user through the user-input devices 106 and the user interface 124 (both of Fig. 1). The system 100 can receive the data character-by-character, when the data-entry field is full, or when the user attempts to continue, such as by tabbing to move to another data-entry field.

The real-time validation tool 122 can validate the data in real-time, such as when it is entered, a user attempts to continue to another data-entry field, presses enter, the data-entry field is full, or the like.

In one implementation, however, the real-time validation tool 122 does not validate data in real-time, instead waiting to validate data until multiple data-entry fields of an electronic form have had data entered into them. One example of this implementation is when the real-time validation tool 122 waits to validate data entered into an electronic form until the user attempts to save or submit the electronic form.

In the foregoing example, the system 100 receives "1/27/2002" from the user when the user attempts to advance to the next data-entry field.

At block 308, the system 100 validates the data received into the data-entry field in the electronic form. The system 100, through the real-time validation tool 122, analyzes the data to determine if it is valid. The real-time validation tool 122 refers to validation rules, if any, governing that particular data-entry field (in this example the report date data-entry field 202). The real-time validation tool 122 validates the data entered into a data-entry field without the user having to save or submit the electronic form. (It can also, in one implementation, validate data after a user saves or submits the electronic form). The real-time validation tool 122 can validate the data entered by applying validation rules associated with the node of the structured data file corresponding to data-entry field into which the data was entered.

The real-time validation tool 122 can apply validation rules from many different sources. One source for validation rules is a schema governing the

structured data file. Other sources of validation rules can include preset and script-based custom validation rules.

Schema-based, preset, script-based, and other types of validation rules can be used to validate data through XML elements when a structured data file includes XML. Use of XML elements in validating structured data files will be discussed following the discussion of the process 300.

For script-based custom validation rules, the real-time validation tool 122 enables these rules to refer to multiple nodes in a structured data file, including nodes governing or governed by other nodes. Thus, the real-time validation tool 122 can validate data from a data-entry field intended for a particular node by checking validation rules associated with that particular node. Through so doing, the real-time validation tool 122 can validate data entered into one node of a group with the validation rules governing the group of which the node is a part. For example, if a group of nodes contains four nodes, and is associated with a script-based validation rule requiring that the total for the data in all of the four nodes not exceed 1000, the real-time validation tool 122 can validate each node against this rule. Thus, if the first node contains 100, the second 400, and the third 300, the real-time validation tool 122 will find the data intended for the fourth node invalid if it is greater than 200 (because 100+400+300+200=1000). Custom script-based validation rules and preset validation rules will be discussed in greater detail below.

In some cases the real-time validation tool 122 can build validation rules from a schema containing logic that governs a structured data file. This logic sets forth the bounds of what data nodes in a structured data file can contain, or the structure the nodes should have. Data entered into a structured data file can

violate this logic, making the structured data file invalid. This invalid data may cause a structural error or a data-type error in the structured data file, possibly making the structured data file useless. To combat this, the real-time validation tool 122 can build validation rules from a structured data file's schema.

Because structural errors are especially important, the real-time validation tool 122 treats these types of errors seriously. To make sure that a user treats these errors seriously, the real-time validation tool 122 builds validation rules for structural errors that stop a user from continuing to edit an electronic form if the real-time validation tool 122 detects a structural error. Validation rules that stop the user from continuing to edit the electronic form (except for fixing that invalid data) are called modal validation rules, and errors that violate them, modal errors.

For less serious errors, such as data-type errors, the real-time validation tool 122 builds validation rules that do not stop the user from continuing. These are called modeless validation rules, and errors that violate them, modeless errors. Modal and modeless validation rules and errors will be discussed in greater detail below.

To aid the real-time validation tool 122 in validating data in real-time, validation rules are associated with particular nodes. By so doing, with each new piece of data received, the real-time validation tool 122 is capable of comparing the data received against an appropriate list of validation rules associated with the node for which the data received is intended. Because this list of validation rules can be very short for each particular node, the real-time validation tool 122 has fewer validation rules to check for each piece of data entered than if it had to check all the validation rules for the node's structured data file. This speeds up the process of validation.

Validation rules, when applied to a structured data file including XML, can be associated with particular nodes using XPath expressions. XPath is a language that describes a way to locate and process items in structured data files by using an addressing syntax based on a path through the file's logical structure or hierarchy. XPath is specified as part of both XSLT and XPointer (SML Pointer Language). It can be used in XML or other contexts. Use of XPath expressions to associate particular nodes with particular validation rules will be discussed following the discussion of the process 300.

Continuing the previous example, at the block 308 the system validates the data entered, "1/27/2002", against validation rules associated with the report date data-entry field 202, thereby determining if the data entered is valid.

In block 310 the system 100 determines whether to proceed to block 314 or 312 depending on whether the data is valid. If the real-time validation tool 122 determines that the data entered is not valid, it proceeds to the block 314, discussed below. If, on the other hand, the real-time validation tool 122 determines it to be valid, the system 100 continues to block 312, allowing the user to continue editing the electronic form. Continuing the ongoing example, if the real-time validation tool 122 determines that the data "1/27/2002" is valid, the system 100 continues on to the block 312. If not, it proceeds to block 314.

At the block 312, the system 100 enables the user to input data into another data-entry field. In Fig. 2, for example, it would allow the user to proceed to enter data into the expense code data-entry field 204 after the data entered into the report date data-entry field 202 was determined to be valid. The system 100 can allow the user to proceed to another data-entry field as well, depending on the user's preference.

If the data is invalid, the system 100 proceeds to the block 314. At the block 314 the system 100, through the real-time validation tool 122, determines whether to proceed to block 316 if the error is not modal and 318 if it is.

Continuing the previous example, assume that the data entered into the report date data-entry field 202 is invalid. Assume also that "1/27/2002" is not defined to be a modal error. (Modal errors are those for which the real-time validation tool 122 rolls back the invalid entry requiring the user to re-enter another entry before continuing on to edit another data-entry field or requires the user to correct.) Thus, in this example, "1/27/2002", is invalid, but is a modeless error.

In the block 316, the real-time validation tool 122 alerts the user of a modeless error by marking the data-entry field as containing an error, but allows the user to continue editing the electronic form. To make the editing process as easy, intuitive, and efficient as possible, the real-time validation tool 122 can mark the data-entry field from which the invalid error was entered in many helpful ways. The real-time validation tool 122 can highlight the error in the data-entry field, including with a red box, a dashed red box, a colored underline, a squiggly underline, shading, and the like. The real-time validation tool 122 can also alert the user with a dialog box in a pop-up window, either automatically or only if the user asks for information about the error.

The real-time validation tool 122, for example, can present a dialog box or other presentation manner explaining the error or what type of data is required by the data-entry field. The real-time validation tool 122 can present a short comment that disappears quickly or is only shown if the user moves his cursor or mouse pointer over the data-entry field. The real-time validation tool 122 can also

provide additional information on request. Many manners of showing the user that the data is invalid as well as showing information about the error can be used. These ways of notifying the user can be chosen by a developer when creating a custom validation rule, which will be discussed in greater detail below.

Fig. 5, for example, shows one manner in which the real-time validation tool 122 can notify the user of an error. In Fig. 5, the expense report 201 electronic form shows that the data entered into the report date data-entry field 202 is invalid with a dashed, red-lined box surrounding the report date data-entry field 202 (visible as a dashed, gray-lined box).

Fig. 6 shows another example. Here, the expense report 201 electronic form shows that the data entered into the report date data-entry field 202 is invalid with a dialog box 602. This dialog box can pop up automatically or after the user requests information, such as by moving his mouse pointer onto the report date data-entry field 202. Also in this example, Fig. 6 shows an option for the user to gain additional information about the error and/or data-entry field by selecting an auxiliary information option 604 entitled "full error description". If the user selects this option, the system 100 will present the user with more information about the error and/or what the data-entry field requires (not shown). The real-time validation tool 122 allows the user to select additional information through a tool-tips icon, right-clicking on the data-entry field, and menu commands for navigating errors. It can also present additional information for multiple errors at once, such as through a list presenting information about every error in an electronic form.

Returning to the dialog box 602, it contains error information 606. This error information 606 reads: "The report date occurs before the end of the expense

period." This informs the user that the data entered, "1/27/2002" is invalid because it violates a rule requiring the report date to occur after the expense period, shown in an expense period data-entry field 608 as "2/3/2003".

In some cases, if the real-time validation tool 122 determines that data entered in a data-entry field is invalid, it will mark other data-entry fields. This is because another data-entry field may actually contain the invalid data. In Fig. 6, for example, the real-time validation tool 122 marked the data entered into the report date data-entry field 202 ("1/27/2002") as invalid because it was prior to the date entered into the expense period data-entry field 608 ("2/3/2003"). The data entered into the expense period data-entry field 608 may actually be the date in error, however, rather than that entered into the expense report data-entry field 202. In these types of cases, the real-time validation tool 122 can mark both fields (not shown).

The real-time validation tool 122 can mark either data-entry field in the above-disclosed manners. It can, for example, mark the report date data-entry field 202 with a dashed red-lined box (shown in Fig. 6) and the expense period data-entry field 608 with a solid-red-line box (not shown). The real-time validation tool 122 can also mark a data-entry field that is invalid because invalid data has been entered into it, with a dashed red-lined box and a data entry field that is invalid because it does not contain any data, with a solid, red underline. In this implementation the real-time validation tool 122 marks the data-entry fields differently so that the user knows quickly and easily that each of these data-entry fields needs to be investigated, but can differentiate between them.

For these modeless errors, the real-time validation tool 122 permits the user to proceed, according to the block 312, discussed above.

For modal errors, however, the real-time validation tool 122 presents a dialog (block 318). The user then can dismiss the dialog. Once the dialog is dismissed, the real-time validation tool 122 rolls back the invalid entry and enables the user to continue editing the electronic form. This editing can include re-inputting data into the data-entry field (block 320), or editing another data-entry field. Alternatively, the real-time validation tool 122 leaves the error in the document, but will not allow the user to continue editing the document without first correcting the error.

In the block 318, the real-time validation tool 122 presents an alert to notify the user of the invalid entry. This alert is intended to inform the user that the error is important and must be fixed. It does not have to be a pop-up window, but should be obvious enough to provide the user with an easy-to-notice notification that the user has entered data causing an error. The alert, in one implementation, is a pop-up window that requires the user to pause in editing the electronic form by making the user click on an "OK" button in the alert. This stops the user mentally, helping the user to notice that he must fix the data-entry field having the error before proceeding. The alert can contain no, little, or extensive information about the error. The information can be presented automatically or after the system 100 receives a request for the information.

Fig. 7 shows the partially filled-in expense report 201 electronic form with a date dialog box 702 arising from invalid data causing a modal error. The dialog box contains a button marked "OK" that the user must select (a date dialog button 704). The date dialog box 702 also contains a date information line 706 informing the user about the error, "The Report Date Must Be Later Than the Expense

Period." This information is intended to aid the user's attempt to correct the invalid data.

Fig. 8 shows another example of a dialog box used for a modal error. In Fig. 8, a nearly all-filled-in expense report 201 electronic form with an invalid number dialog box 802 is shown. This expense report 201 contains many filled-in data-entry fields, each of which is not shown to be invalid with the exception of a cost data-entry field 804. The cost data-entry field 804 contains a modal error, "a". When the user entered the textual data "a", the real-time validation tool 122 found it invalid and presented the invalid number dialog box 802. The invalid number dialog box 802 informs the user through an invalid number information line 806 that the data entered is not valid because it is not a number between negative and positive 1.7976913486231E308. In this example it is not a valid number because it is not a number at all. Like the prior example, the user must select a button in the dialog box, here an invalid number button 808.

After presenting the user with some sort of alert in block 318 (Fig. 3), the real-time validation tool enables the user to re-input data into the data-entry field containing the modal error (block 320). Here the user must change the data within the data-entry field to a valid or modeless error before continuing to edit new data-entry fields in the electronic form. Once the user inputs new (or the same) data into the data-entry field (such as the cost data-entry field 804 of Fig. 8), the system 100 receives the data at the block 306 and so forth. To proceed, the user must enter data that is not a modal error; if the user does not, the system 100 will follow the process 300, continuing to find the data modally invalid and not permit the user to continue.

Through this process 300 of Fig. 3, the system 100 can receive and validate data in real-time. By so doing, a user can easily, accurately, and efficiently edit a structured data file through entry of data into data-entry fields in an electronic form.

The examples set forth in Figs. 2 and 4-8 are examples, and are not intended to be limiting on the abilities of the system 100 or the real-time validation tool 122; other types of forms, data-entry fields, and alerts can be used.

Validating Data Using XML Elements

Overview

As set forth above, the real-time validation tool 122 is capable of validating data received into a data-entry field in an electronic form. To do so, the real-time validation tool 122 uses validation rules, if any, associated with the particular data-entry field and its corresponding node. The real-time validation tool 122 can validate the data entered by applying the validation rules associated with the node of the structured data file corresponding to data-entry field into which the data was entered.

In an implementation set forth below, the real-time validation tool 122 validates structured data files that include XML. In this implementation, the real-time validation tool 122 uses XML elements to aid it in validating nodes of an XML data file.

Error-Condition XML Elements

An error-condition XML element is one of various types of XML elements that can be used by the real-time validation tool 122 in validating data within a

node of an XML data file. One way in which the real-time validation tool 122 can use this XML element to validate data within nodes of an XML data file is set forth in an exemplary process below.

Fig. 9 shows an exemplary process 900 showing how the real-time validation tool 122 can use XML elements to validate data in an XML data file. The process 900 and other processes that follow are illustrated as a series of blocks representing individual operations or acts performed by the system 100. These processes may be implemented in any suitable hardware, software, firmware, or combination thereof. In the case of software and firmware, these processes represent a set of operations implemented as computer-executable instructions stored in memory (such as in one or more computer readable media) and executable by one or more processors.

Some of the blocks of the process 900 can be performed at various points before, after, or parallel with the blocks of Fig. 3.

In one implementation, various blocks of the process 900 are an exemplary implementation of block 308 of Fig. 3, and to a limited extent blocks 310, 314, 316, and 318, also of Fig. 3. In this implementation, the real-time validation tool 122 is validating data in data-entry fields in real-time. Because of this, the real-time validation tool 122 associates one or more error-condition elements to nodes of the XML data file prior to receiving the data at block 306. This and other nodes of the XML data file can be associated with one or more error-condition XML elements prior to receiving the data entered into the data-entry field. In these cases, the system 100 performs blocks 902 through 906, 910, or 912 prior to attempting to validate the data for that particular node. Thus, in this implementation the node corresponding to the data-entry field in which the data is

entered is associated (if appropriate) with the error-condition XML element prior to receipt of the data in block 306 of Fig. 3.

In some cases, however, the real-time validation tool 122 is validating data from multiple data-entry fields and not in real-time (such as when the real-time validation tool 122 waits to validate data until a user saves or submits data entered into an electronic form). In these cases, the real-time validation tool 122 can associate the node or nodes to be validated with the error-condition elements prior to, incident with, or after receipt of data into data-entry fields of the electronic form that corresponds to the XML data file.

By way of review, at block 306 of Fig. 3, the system 100 receives the data entered into the data-entry field by the user prior to validating that data in block 308, which the system 100 then validates in real time.

At block 902, the real-time validation tool 122 reads an error-condition XML element.

Fig. 10 sets forth an error-condition XML element 1002. This error-condition XML element 1002 includes an error-condition match attribute 1004 and a Boolean expression attribute 1006. The error-condition match attribute 1004 identifies nodes of an XML data file on which the error-condition XML element 1002 is declared. The Boolean expression attribute 1006 is evaluated to validate any XML nodes of the XML data file identified by the error-condition match attribute 1004.

The error-condition XML element 1002 can also include an expression-context attribute 1008 and a show-error-location attribute 1010, which are described below.

Thus, at block 902, the real-time validation tool 122 reads at least the error-condition match attribute 1004. The real-time validation tool 122 can also read the other attributes at this block 1002, or can wait to do so if or until the information in each attribute is needed.

At block 904, the real-time validation tool 122 parses the error-condition match attribute 1004 into an XPath expression. As shown in Fig. 10, the error-condition match attribute 1004 includes an error-condition match syntax 1012 and an error-condition XPath expression 1014. Thus, at block 904, the real-time validation tool 122 gains the XPath expression 1014.

The error-condition XML element 1002 can be, for instance:

```
<xsf:errorCondition match="/exp:expenseReport"
                    expressionContext="exp:reportDate"
                    expression="msxsl:string-compare
                        (., ../exp:startDate) < 0 and
../exp:startDate != ''''>
    ...
    </xsf:errorCondition>
```

In this example, the match syntax 1012 is " errorCondition match= " and the XPath expression 1014 is " "/exp:expenseReport" ".

At block 906, the real-time validation tool 122 finds nodes of the XML data file (or compares the node being validated) that match the match pattern using the XPath expression 1014. Thus, those nodes of the XML data file that match the error-condition XPath expression 1014 are subject to being found valid or invalid

based on whether the data in the node is found to violate the Boolean expression attribute 1006 (as discussed below). In the ongoing example, nodes of the XML data file named "exp:expenseReport" will be found by the match pattern. For clarity in the ongoing discussion, these nodes or node will be referred to as "found nodes."

At block 908, the real-time validation tool 122 can parse the expression-context attribute 1008. If the real-time validation tool 122 has not previously read this attribute 1008 from the error-condition XML element 1002, the real-time validation tool 122 does so prior to or as part of block 908.

The real-time validation tool 122 parses the expression-context attribute 1008 to obtain an expression-context syntax 1016 and an expression-context XPath expression 1018. This expression-context XPath expression 1018 specifies nodes that should be evaluated in addition to the found nodes. These additional nodes (called "the set of nodes") are nodes related to the found nodes. Thus, at block 908, the real-time validation tool 122 gains the expression-context XPath expression 1018.

At block 910, the real-time validation tool 122 can use the expression-condition XPath expression 1018 parsed in block 908 to obtain the set of nodes from the found nodes. This set of nodes, if non-zero, includes nodes in addition to the found nodes.

At block 912, the real-time validation tool 122 parses the Boolean expression 1006. The Boolean expression 1006 includes a Boolean syntax 1020 and a Boolean XPath expression 1022. In the ongoing example, the Boolean syntax 1020 is " expression " and the Boolean XPath expression 1022 is " "msxsl:string-compare(., ../exp:startDate) < 0 and ../exp:startDate !=" ".

At block 914, the real-time validation tool 122 determines whether or not the found node(s) violate the Boolean XPath expression 1022 shown in the Boolean expression 1006. It can do so by waiting until a found node has data entered into a corresponding data-entry field, either in real-time or otherwise. In one implementation, the real-time validation tool 122 also determines whether or not the set of nodes of the found nodes also violate the Boolean XPath expression 1022, either alone or in combination with the found node. Thus, in some cases a node from a set of nodes that relates to a found node can be deemed or marked invalid. This can aid a user in understanding how to fix data that is causing a found node to be invalid.

Continuing the ongoing example, if a found node or a node from the set of nodes includes a date that is earlier than a start date, the node will be found to violate the Boolean XPath expression 1022. And thus, the Boolean XPath expression 1022 will return a TRUE value. A TRUE value indicates to the real-time validation tool 122 that the data in the node in question is not valid.

At block 916, for every node found to violate the Boolean XPath expression, the real-time validation tool 122 associates the violating nodes with an error message. One way in which the real-time validation tool 122 can associate an error message with the invalid node(s) is set forth the process 1100 of Fig. 11.

At block 918, the error message can be displayed. It can be displayed as described in Figs. 3 or 11.

Error-Message XML Elements

An error-message XML element is one of various types of XML elements that can be used by the real-time validation tool 122 as part of validating data

within a node of an XML data file. One way in which the real-time validation tool 122 can use this XML element as part of validating nodes is set forth in an exemplary process below.

Fig. 11 shows an exemplary process 1100, which is an exemplary implementation of blocks 916 and 918 of Fig. 9 and/or blocks 1312 and 1314 of Fig. 13 (set forth below). Process 1100 sets forth a process for associating an invalid node with an error message using an XML element. The process 1100 also shows how the system 100 can display that error message.

At block 1102, the real-time validation tool 122 determines that a node of an XML data file is associated with an error-message XML element. A node of an XML data file can be associated with an error-message XML element when that node is deemed invalid. As set forth above, the real-time validation tool 122 can determine that one or more nodes in an XML data file violate a validation rule. In process 900 at block 914, for instance, the real-time validation tool 122 can determine that certain nodes violate the Boolean XPath expression 1022 of the error-condition XML element 1002. By so doing, those nodes are deemed invalid.

At the block 1102, the real-time validation tool 122 determines which nodes are invalid and which validation rule the node violates. It is useful for a user attempting to fix invalid data that each particular validation rule has a particular error message. That way, the user is more likely to understand what is wrong with the data.

For example, if data in a node is " $531.00 ", and a validation rule requires that the data in that node not be less than zero or more than 500 dollars, the validation rule is violated. That rule can have a particular error message, such as

"Value must be between zero and 500 dollars." With this error message a user can quickly understand that the data entered is too large a dollar amount.

Also for example, the rule violated in Fig. 6 has a particular error message, there referred to as the error information 606. That error message is " The report data occurs before the end of the expense period. ".

Based on the relationship between a validation rule and its error message, the real-time validation tool 122 can determine which error message is associated with the invalid node. In one implementation, the violated rule, such as the error-condition XML element 1002, contains information aiding the real-time validation tool 122 in determining that a particular error-message XML element should be associated with the invalid node.

Fig. 12 sets forth an error-message XML element 1202. This error-message XML element 1202 includes an error message attribute 1204. It can also include an error-type attribute 1206. The error-type attribute 1206 can include either a modal-error-type XPath expression 1208 or a modeless-error-type XPath expression 1210.

At block 1104, the real-time validation tool 122 reads the error-message attribute 1204 from the error-message XML element 1202.

At block 1106, the real-time validation tool 122 associates the error message read from the error-message attribute 1204 with the invalid node.

At block 1108, the real-time validation tool 122 reads the error-type attribute 1206 from the error-message XML element 1202.

At this point, the real-time validation tool 122 has associated an error message with an invalid node and has found what type of error the invalid node is

committing. The type of error being set forth is either the modal-error-type XPath expression 1208 or the modeless-error-type XPath expression 1210.

At block 1110, the real-time validation tool 122 proceeds along the "No" path to block 1112 if the error-type is modeless (the modeless-error-type XPath expression 1210) and along the "Yes" path if it is modal (the modal-error-type XPath expression 1208).

At block 1112, the system 100 displays or makes available the error message from the error-message attribute 1204. If the system 100 makes the error message available but does not display it, the system 100 indicates that the data in the data-entry field associated with the node is invalid. It can do so, as set forth above, by surround the data-entry field with a dashed, red line, for instance. The user can then select that the error message be displayed, such as by right-clicking on the data-entry field with the mouse 114 of Fig. 1.

In one implementation, the system 100 displays the error message at certain locations on an electronic form shown on the screen 104. In this implementation, the real-time validation tool 122 reads the show-error-location attribute 1010 of the error-condition XML element 1002.

This attribute shows where an error-message should be shown on an electronic form to which the invalid node is associated. In certain cases, for instance, the show-error-location attribute 1010 includes instructions as to which data-entry field of the electronic form the message should be next to or selectable from. If an invalid node is associated with a parent or root node, such as through the expression-context attribute 1008, it can be appropriate to display the error message next to the data-entry field of the parent or root node, rather than the invalid node. One such example of this is where the user can most easily fix an

invalid entry through a data-entry field of the parent or root node of the invalid node, rather than the data-entry field in which the user entered the data causing the invalid state.

At block 1114, the real-time validation tool 122 rolls back data in the invalid node (and its associated data-entry field). In one implementation, the error-message XML element 1202 includes a long error-message attribute (not shown).

At block 1116, if the error-message element 1202 includes the long error-message attribute, the real-time validation tool 122 reads this attribute.

At block 1118, the system 100 displays the error message(s). If a long error-message attribute is included in the error-message XML element 1202, the system 100 displays or makes available for display both the long message of that attribute and the shorter message. If it does not, the system 100 displays the error message (here the shorter message) from the error-message attribute 1204.

Either of these messages can be displayed at a particular location of the electronic form on the display 104, as set forth in the implementation described as part of the block 1112.

Override XML Elements

An override XML element is one of various types of XML elements that can be used by the real-time validation tool 122 to aid a user in correcting invalid data within a node of an XML data file. One way in which the real-time validation tool 122 can use this override XML element is to provide an error message to a user for a node that violates a schema-based validation rule.

Fig. 13 shows an exemplary process 1300 showing how the real-time validation tool 122 can use an override XML element to aid a user in correcting invalid data in an XML data file.

Some of the blocks of the process 1300 can be performed at various points before, after, or parallel with the blocks of Fig. 3.

In one implementation, various blocks of the process 1300 are an exemplary implementation of blocks 308 and 318 of Fig. 3. The real-time validation tool 122 can perform any of the blocks 1302 through 1306 prior to or incident with block 308 of Fig. 3. Thus, a node being validated, whether in real-time or otherwise, can be associated (if appropriate) with the override XML element prior to, incident with, or after receipt of the data for that node in block 306 of Fig. 3.

At block 1302, the real-time validation tool 122 reads an override XML element for an override match pattern attribute.

At block 1304, the real-time validation tool 122 parses the override match attribute into an XPath expression. The override XML element can be, for instance:

```
<xsf:override  match="Amount" >
    ...
</xsf:override>
```

In this example, the override XML element includes an override match attribute having an override match syntax of " override " and an override match XPath expression of " "Amount" ".

At block 1306, the real-time validation tool 122 determines which node of the XML data file (or whether or not the node in which data was just entered) matches the match pattern using the override XPath expression. In this example, only a node named "Amount" in an XML data file would match. The matching node will be referred to as the "found node".

At block 1308, the real-time validation tool 122 determines whether or not the found node violates a schema of the XML data file. This can be performed in various manners as set forth herein for validating a node of a structured data file using its schema.

At block 1310, the real-time validation tool 122 reads an error-message XML element associated with the override XML element. In one implementation this error-message XML element is the error-message XML element 1202.

At block 1312, the real-time validation tool 122 associates the error message from the error-message XML element with the found node.

At block 1314, the system 100 can display the error message. It can be displayed as described in Figs. 3 or 11.

Event-Handler XML Elements

An event-handler XML element is one of various types of XML elements that can be used by the real-time validation tool 122 to validate and/or aid a user in correcting invalid data within a node of an XML data file. One way in which the real-time validation tool 122 can use this event-handler XML element is to execute code when a node of the XML data file is modified. On execution, the code can perform various different operations, such as reject data entered into a node, modify the data in the node or other nodes, or modify files associated with the

XML data file, for instance.

Fig. 14 shows an exemplary process 1400 showing how the real-time validation tool 122 can use an event-handler XML element as an aid in validating nodes of an XML data file and performing many other operations.

Some of the blocks of the process 1400 can be performed at various points before, after, or parallel with the blocks of Fig. 3.

In one implementation, various blocks of the process 1400 are an exemplary implementation of blocks 308 through 320 of Fig. 3. The real-time validation tool 122 can perform any of the blocks 1402 through 1412 prior to or incident with block 308 of Fig. 3. Thus, a node being validated, whether in real-time or otherwise, can be associated (if appropriate) with the event-handler XML element (or its handler-object name) prior to, incident with, or after receipt of the data for that node in block 306 of Fig. 3.

At block 1402, the real-time validation tool 122 reads an event-handler XML element.

Fig. 15 sets forth an event-handler XML element 1502. This event-handler XML element 1502 includes an event-handler match attribute 1504 and a handler-object attribute 1506. The event-handler match attribute 1504 identifies nodes of an XML data file on which the event-handler XML element 1502 is declared. The handler-object attribute 1506 identifies executable code that is capable of being called when an identified node of an XML data file is modified.

The handler-object attribute 1506 includes a handler-object syntax 1508 and a unique hander-object name 1510, which will be discussed below.

Thus, at block 1402, the real-time validation tool 122 reads at least the event-handler match attribute 1504. The real-time validation tool 122 can also

read the handler-object attribute 1506 at this block 1402, or can wait to do so if or until needed.

At block 1404, the real-time validation tool 122 parses the event-handler match attribute 1504 into an XPath expression. As shown in Fig. 15, the event-handler match attribute 1504 includes an event-handler match syntax 1512 and an event-handler XPath expression 1514. Thus, at block 1404, the real-time validation tool 122 gains the XPath expression 1514.

The event-handler XML element 1502 can be, for instance:

```
<xsf:domEventHandler  match="TravelReport/Expenses"
                      handlerObject="TravelExpenses" />
```

In this example, the match syntax 1512 is " domEventHandler match= " and the XPath expression 1514 is " "TravelReport/Expenses" ".

At block 1406, the real-time validation tool 122 parses the handler-object attribute 1506 for the handler-object name 1510. In the ongoing example, the handler-object name 1510 is " "TravelExpenses" " and the handler-object syntax 1508 is " handlerObject= ".

At block 1408, the real-time validation tool 122 creates a handler-object that references executable code and has the name given in the handler-object name 1510.

At block 1410, the real-time validation tool 122 finds nodes of the XML data file (or compares the node in which data was just entered) that match the match pattern using the XPath expression 1514. Thus, when those nodes of the XML data file that match the event-handler XPath expression 1514 are modified,

the real-time validation tool 122 can execute the code referenced by the handler-object with the handler-object name 1510.

At block 1412, the real-time validation tool 122 associates the found nodes (or the node just modified) with the handler-object name 1510. By so doing, when the real-time validation tool 122 determines that a node has been modified (including by being added or deleted from the XML data file), the real-time validation tool 122 can simply execute the executable code having the event object with the handler-object name 1510.

At block 1414, the real-time validation tool 122 executes the code that has the event object with the handler-object name 1510.

At this point, the executable code can perform many different operations. The code can be of many different types and can, for instance, validate or invalidate the node.

In one implementation the code is custom script-based code set forth herein. This custom script-based code can include, for instance, the script 2502 set forth in Fig. 25 (described below).

The event-handler XML element 1502 can show the real-time validation tool 122 when the executable code is to be executed.

In one implementation, for example, the real-time validation tool 122 executes the code prior to accepting data entered into a node associated with the handler-object name 1510 and letting a user continue to edit the XML data file. An example of this is set forth in Fig. 3 at block 314, where the real-time validation tool 122 rejects the data because it is deemed a modal error (it then presents instructions to the user to re-enter data into that node).

This is described in greater detail as part of the description relating to Figs. 24 and 25.

Creating Custom Validation Rules For Structured Data Files

Overview

The system 100 of Fig. 1 includes the real-time validation tool 122, which enables a developer to create or customize validation rules for a structured data file. To enable the developer to choose a node for which to create or customize a validation rule, the real-time validation tool 122 displays nodes of a structured data file or its generalized instance and/or the data-entry fields mapped to those nodes. A generalized instance is a structured data file that has been generated from a schema and is comprehensive enough to illustrate all structural patterns allowed by the schema. Nodes in the generalized instance are a coupling of a node from a structured data file and a part of the structured data file's schema that governs that node. (For more on nodes, see the description relating to Fig. 21, below.) Because the nodes of a generalized instance for a structured data file are related to the nodes of the structured data file, nodes of the generalized instance can be chosen in place of a related node of a structured data file. In addition, data-entry fields that map to either of those nodes can also be selected in their place. In each of these cases, the real-time validation tool 122 recognizes the node of the structured data file to which a validation rule should be applied.

If the electronic form is in the process of being built, the developer will probably find it easiest to add custom validation rules while creating the form. Because of this, the real-time validation tool 122 enables the developer to add a

custom validation rule to a data-entry field as that data-entry field is being added to the electronic form.

The real-time validation tool 122 also enables a developer to add custom validation rules to nodes after an electronic form mapped to the structured data file (or its generalized instance) has been created. In either case, the real-time validation tool 122 enables a developer to easily create custom validation rules for nodes of a structured data file, thereby improving a data-entry user's ability to accurately and efficiently enter data for storage in a structured data file. This is because when a user later enters data intended for that particular node, the real-time validation tool 122 can access the custom validation rule for that node. Custom validation rules make editing a structured data file more accurate and efficient.

In addition, the real-time validation tool 122 enables a developer to create custom validation rules in an easy-to-use way by allowing the developer to choose from preset validation rules. These preset validation rules can be chosen quickly and easily by a developer even if the developer is not skilled in programming.

Fig. 16 shows a process 1600 for adding validation rules for nodes in a structured data file. The process 1600 is illustrated as a series of blocks representing individual operations or acts performed by the system 100.

Selecting a Node and a Preset or Script-Based Validation Rule

At block 1602, the real-time validation tool 122 enables a developer to select a node in a generalize instance or structured data file either directly or through selecting a data-entry field in an electronic form that is associated with that node. A developer can select a node by right-clicking on it with the mouse

114, entering a command in the keyboard 110, or in some other manner through the other device(s) 112 (all of Fig. 1).

Fig. 17 shows an exemplary custom validation screen 1700, including a development copy of an expense report electronic form 1702. The electronic form 1702 is a development copy because it is a copy that is not intended for data entry by a user, but rather editing of its form and structure by a developer. In this example, each data-entry field shown in the electronic form 1702 is associated with a node in a generalized instance and/or structured data file. The development electronic form 1702 is displayed by the real-time validation tool 122 to enable a developer to select a data-entry field associated with a node. By doing so, the developer can add a custom validation rule to that node.

At block 1604, the system 100 receives a selection of a node (directly or through selection of a data-entry field).

Fig. 17 shows an example of a node chosen by a developer. In this example, the developer selected a date data-entry field 1704. Once received, the system 100 reacts according to block 1606.

At the block 1606, the system 100 enables a developer to select a preset or script-based validation rule. The system 100 can enable the developer's selection through many user-interface manners, including by presenting a pop-up window with various options, one of which includes an option to add a custom validation rule to the selected node. The developer can choose from a preset list of validation rules or can choose to create his or her own validation rule by creating script.

Fig. 17 shows a properties box 1706, providing the developer with an option to perform various functions, one of which includes an option to customize the validation rules for the selected node. By clicking on or otherwise selecting a

data validation option button 1708, the developer can choose to see a validation option box, shown in Fig. 18.

Fig. 18 shows an exemplary custom validation screen 1800, including the development copy of the expense report electronic form 1702 and a validation option box 1802 entitled "Data Validation (Date)". In this example, the "(Date)" part of the title is from the name for the selected data-entry field in the electronic form 1702. This feature is intended to make it easier for the developer to keep track of the node for which he is adding a validation rule.

The validation option box 1802 is used to enable the developer to choose which type of custom validation rule to add (and/or edit, if one already exists). The developer can choose to add a preset custom validation rule by selecting an add preset rule button 1804. The developer can also choose to add a script-based validation rule by selecting either of two events in an event box 1806, OnBeforeChange event 1808 or OnValidate event 1810. Script-based validation rules and events used in them will be discussed in greater detail below.

At block 1608, the system 100 determines whether the developer selected an option to add a custom validation rule using preset rules or script-based rules. If the developer chose preset rules, the system 100 proceeds to block 1610. If the developer chooses to create a script-based validation rule, the system 100 proceeds to block 1612.

Preset Validation Rules

At the block 1610, the system 100 enables selection of preset rules. The system 100 enables a developer to select from a list of many validation rules that are desirable for validating data. These preset validation rules can be selected by

the developer in an easy, efficient manner. Also, these preset validation rules enable the developer to create powerful validation rules for the real-time validation tool 122 to use when validating data. Another benefit of these preset validation rules is that the developer does not need to know how to program or write code (script or otherwise). Also, these validation rules do not have to be exclusive, they can be added to other validation rules, such as validation rules based on the structured data file's schema or custom script-based validation rules. Thus, these preset validation rules can allow a developer with little or no programming ability to create a broad range of useful validation rules, making it accurate and efficient for an eventual user to edit a structured or unstructured data file.

Many different types of preset validation rules can be made available by the system 100. These can include rules that require data entered to be of a certain size, be numbers or text, and compare in certain ways with data from other data-entry fields, for example.

When used to validate XML data files, these present validation rules can be used to generate rules represented by XML elements.

To give a developer flexibility, the preset validation rules can be adjusted by the developer entering numbers or text, or relating data in one field to another. Examples of how this can be done will be discussed in the following example in Fig. 19.

Fig. 19 shows an exemplary custom validation screen 1900, including the development copy of the expense report electronic form 1702, the properties box 1706 (obscured), the validation option box 1802 (obscured), and a preset validation selection box 1902, entitled "Preset Validation (Date)". In this

example, the "(Date)" part of the title is from the name for the selected data-entry field in the electronic form 1702. This feature is intended to make it easier for the developer to keep track of the node for which he is adding a validation rule.

In this example, the developer chose to add a preset validation rule by selecting the add validation button 1804 of Fig. 18 for a node corresponding to the date data-entry field 1704.

Once that selection was received by the system 100, the system 100 presented the preset validation selection box 1902. In this example, the developer has selected to add a preset validation rule and is attempting to view the different options for preset validation rules. Thus, the system 100 is displaying a list of preset validation rules through a preset validation list 1904 in Fig. 19.

From this list the developer can create a validation rule. The developer can choose to require (via the validation rule) that data entered into the date data-entry field 1704 be of a certain sort. Using the preset validation list 1904 as an example, the developer can choose a particular type of preset validation rule.

With the preset validation rule selected, the developer can then enter text, numbers, another node, or whatever is appropriate. The developer can select a preset validation rule and then add, into a validation field 1906, numbers, text, a node, or etc., to complete the validation rule. The system 100 can intelligently aid the developer by providing appropriate options, such as suggesting a date for the date data-entry field 1704. This is another aid to guide the developer, helping him or her to easily add and/or edit validation rules.

The developer can choose from various useful preset validation rules, such as those set forth in Fig. 19 in the preset validation list 1904. This list includes preset validation rules of: "is equal to"; "is not equal to"; "is less than"; "is greater

than"; "is greater than or equal to"; "is present"; "is not present"; "is blank"; "is not blank"; "contains"; "does not contain"; "begins with"; and "does not begin with", for example.

Fig. 20 shows an exemplary custom validation screen 2000, including the development copy of the expense report electronic form 1702, the properties box 1706 (obscured), the validation option box 1802 (obscured), the preset validation selection box 1902, and two validation field options, enter date option 2002 and enter field option 2004.

Once a preset validation rule is selected by the developer, such as the "is equal to" preset validation rule, the developer can enter an appropriate date, such as "3/13/2003" into the enter date option field 2002 or select a field with the enter field option 2004. In the present example, the developer does not select to enter a date, but rather selects a data-entry field compared to which the date must be equal in order for the data entered to be valid.

If the developer chooses to select a field (in this present example by selecting the enter field option 2004), the system 100 enables the developer to choose from nodes and/or data-entry field mapped to those nodes. The system 100 can do so simply by allowing the developer to choose from data-entry fields shown in the electronic form or from a list of nodes in the generalized instance. Because the nodes of the generalized instance and the data-entry fields of the electronic form are related, choosing either the node or the data-entry field associated with the node can be allowed by the system 100. Some developers may be unfamiliar with nodes of a generalized instance and so may feel more comfortable choosing from data-entry fields associated with those nodes. The developer need not know that the data-entry fields are associated with nodes,

however. By so enabling the developer to choose in whichever way he or she is comfortable, the system 100 improves the customization experience of the developer.

Fig. 21 shows an exemplary custom validation screen 2100, including the development copy of the expense report electronic form 1702, the properties box 1706 (obscured), the validation option box 1802 (obscured), the preset validation selection box 1902 (partially obscured), and a node selection box 2102.

Continuing the ongoing example, the developer can choose the enter field option 2004 in Fig. 20. After the developer does so, the system 100 presents nodes of the generalized instance or structured data file that map to or govern the data-entry fields in the electronic form. In this example, the system 100 presents nodes from the generalized instance, shown in the node selection box 2102. This enables the developer to choose a node, such as by selecting a start date node 2104 in the node selection box 2102.

Fig. 21 serves to demonstrate the structured format of nodes in a structured data file, as well as the relationship between nodes and data-entry fields. The node selection box 2102 includes a partial list of the nodes of the structured data file corresponding to the expense report electronic form 201. These nodes include nodes mapped to data-entry fields, such as the start date node 2104, an expense code node 2106, an end date node 2108, a report date node 2110, a purpose node 2112, and a notes node 2114. These nodes are mapped, respectively, to the expense period data-entry field 608, the expense code data-entry field 204, an end date data-entry field 2116, the report date data-entry field 202, a purpose data-entry field 2118, and a notes data-entry field 2120. These nodes also include nodes that contain or govern other nodes, such as a manager node 2122, which

governs the nodes 2104 to 2114. The structured aspect of the structured data file and/or generalized instance is shown here by nodes governing other nodes, and graphically by some nodes being presented in the form of a folder icon and some being indented related to others.

At block 1614, the system 100 receives a selection of a preset rule. The system 100 can receive the selection in various ways, including those set forth for selecting nodes and data-entry fields above. The selection of a preset validation rule may include numerous steps, as shown in the foregoing example.

In the foregoing example, because of the developer's selection of the start date node 2104, the system 100 adds a validation rule requiring that data entered into the date data-entry field 1704 be equal to the date entered or retained in the start date node. This start date node is associated with a start date data-entry field 608, shown in Fig. 21.

In cases where the selected preset validation rule is to be applied to a node of an XML data file, the system 100 can represent this selection with an XPath expression.

The system 100 can build an XPath expression so that the real-time validation tool 122 can later apply the type and content of the preset validation rule chosen by the developer. An example of this type of XPath expression is the Boolean XPath expression 1022 of the Boolean expression attribute 1006 of Fig. 10.

Thus, the Boolean XPath expression 1022 can represent a preset validation rule. In this case, the Boolean XPath expression 1022 can represent a preset validation rule chosen by a developer selecting from the start date preset validation list 1904 of "is less than", the enter field option 2004, and the start date node 2104

in the node selection box 2102. With these selections by the developer, the system 100 can build this validation rule with the Boolean XPath expression 1022, which contains " "msxsl:string-compare(., ../exp:startDate) < 0 and ../exp:startDate != " ".

Alerts for the Preset Validation Rule

At block 1616, the system 100 enables the selection of alert information for the user. Alert information can include any information alerting a user of an error, invalid entry, or an action of the real-time validation tool 122 based on the error or invalid entry. Alerts include graphics and/or text, such as error messages.

Before, after, or as part of a developer adding a preset validation rule, the system 100 enables the developer to add alert information that can be presented to the user if the user violates a preset validation rule. The developer can choose from default information or input custom information. The developer can choose how the alerts and their information and graphics are presented, such as through a dialog box in a pop-up window or a line of text appearing if the user moves a mouse icon over the data-entry field. With or without information, the developer can choose from various graphical aspects to be included in an alert, such as box or dashed-line box around the data-entry field, a squiggly line under the data in the data-entry field, shading of the data-entry field, and in other manners.

Fig. 22 shows an exemplary custom validation screen 2200, including the development copy of the expense report electronic form 1702, the properties box 1706 (obscured), the validation option box 1802 (obscured), the preset validation selection box 1902, and an information alert option box 2202.

In the example shown in Fig. 22, the system 100 enables the developer to choose information to be made available to the user if he enters data violating the associated preset validation rule. Here the developer can choose two sets of information to be presented. The system 100 presents the first set of information as a "screen tip", which arises when the user makes the error, and can be presented automatically or if the user moves a mouse icon or otherwise selects the data-entry field containing the invalid data.

The system 100 presents the second set of information either at the request of the user or automatically, depending on the preference of the developer. The developer can choose to have the second set of information presented automatically and in a dialog box in a pop-up window, for instance. The developer can choose for the dialog box to contain a button, such as a button like the invalid number button 808 of Fig. 8, and that the user must click to continue editing the electronic form. A developer could desire to automatically present a pop-up window so that the user takes special attention to the invalid entry. For errors the developer is not as concerned about or if the developer thinks it more efficient for the user to be able to continue editing the electronic form without the interruption of a pop-up window, the developer can choose to have the information only come up at the user's request.

Fig. 23 shows an exemplary custom validation screen 2300, including the development copy of the expense report electronic form 1702, the properties box 1706 (obscured), the preset validation selection box 1902, and the information alert option box 2202. In this example, which is not based on the date data-entry field 1704 of Fig. 22, a developer has entered two messages using the information alert option box 2202. The first, entered into a screen tip field 2302, is presented

to the user as a screen tip, such as is shown in the error information 606 field of Fig. 6. The second, entered into a message dialog field 2304, is presented to the user as a message in a dialog box if the user requests, such as is shown in the date information line 706 of Fig. 7.

In cases where the alert is for a node of an XML data file, the system 100 can represent the alert information chosen, if text, as an error message using an XML attribute.

The system 100 can build an XML attribute so that the error message chosen can be displayed by the real-time validation tool 122 if appropriate, such as when data in a node is found to be invalid. An example of this type of XML attribute is the error-message attribute 1204 of Fig. 12.

Also, the system 100 can build an XPath expression showing whether or not a developer chose an error to be modal or modeless, which can affect how the real-time validation tool 122 treats the error message for that error. Examples of these types of XPath expressions are the modal-error-type XPath expression 1208 and the modeless-error-type XPath expression 1210, both of Fig. 12.

Associating the Preset Validation Rule with Its Node

At block 1620, the system 100 associates the preset validation rule and its alert information with the selected node. The system 100 associates a preset validation rule and its alert information (and/or non-information alert, if applicable) to the node selected for the validation rule by mapping the preset validation rule to the node.

In cases where the data-entry field selected is associated with a node of an XML data file, this mapping can be accomplished through a declarative syntax,

which can include XPath expressions. Each preset validation rule that governs a node, or group of nodes, can be associated with the node or group of nodes with XPath expressions. The alert information can also be associated with the node or nodes with XPath expressions, either alone or along with the preset validation rule.

The system 100 can build an XPath expression so that the real-time validation tool 122 can later apply a validation rule on the correct node. Examples of these types of XPath expressions include the error-condition XPath expression 1014 and the event-handler XPath expression 1514.

Another example of the declarative syntax the system 100 can use to associate a preset validation rule to its node is shown below:

```
<xsf:validationConstraints>
  <xsf:errorCondition
      match="TravelPlan"
          expression=". &gt; ../endDate"
          expressionContext="startDate"
          showErrorOn=".|../endDate">
    <xsf:errorMessage
          type="Modeless"
          shortMessage="short          error
        message">
          long error message
    </xsf:errorMessage>
  </xsf:errorCondtion>
</xsf:validationConstraints>
```

In this example, a preset validation rule is associated with a node, marked as "TravelPlan", of a structured data file. When a user later enters data into a data-entry field mapped to this node, the real-time validation tool 122 can validate the data against the preset validation rule. The preset validation rule shown here is specified by the "expression" attribute and is positive (violated) when the data entered into a data-entry field marked "endDate" violates a condition where the data entered previously into a "startDate" data-entry field is greater than the data entered into the "endDate" data-entry field. The error is shown on the "endDate" data-entry field through the code "showErrorOn=".|../endDate">".

As part of this block 1620, the system 100 can build a file containing the preset validation rules created for the structured data file (or, in some cases, a generalized instance for the structured data file). This file of preset validations can be added to the structured data file's solution. The structured data file's solution, as mentioned above, contains various files, such as a viewing file to create an electronic form.

Returning to the block 1608, if a developer chooses to add a custom validation rule using script, the system proceeds to the block 1612.

Script-Based Validation Rules

At the block 1612, the system 100 enables input of a script-based rule. The system 100 can enable input of a script-based rule in various ways, including easy-to-use ways like presenting a screen for inputting script, providing much of the surrounding script so that the developer does not have to write as much script, and the like. By so doing, the system 100 provides an easy-to-use way for a developer to input validation rules.

The system 100 provides this way of adding validation rules (and related alert information, also through the script) for developers desiring greater control than the preset rules allow, such as a developer wishing to add a validation rule of considerable complexity. Through script the developer can, for example, add a validation rule that compares data received against an entry in a non-local database, such as zip-code data received against postal zip-code database for the United States.

In addition, through script a developer has a lot of flexibility. Script allows a developer to display alert messages in a pop-up window, with a passive screen tip, and in other manners. Script also allows a developer to choose what information is presented to a user and when it is presented. A developer could, for example, have an alert message appear when a user enters invalid data but before the user continues on to another data-entry field. The developer could have an alert appear on the screen with dialog, an alarm and information presented through audio (if the system 100 connects to speakers), and/or have the data-entry field with the invalid data highlighted in various ways. The developer could choose for the data-entry field to be highlighted with a box, a dashed-line box, shading, underlining, and choose the color for each.

Script also allows a developer to set whether or not the validation rule, when violated, results in a modal or modeless error. By so doing, the real-time validation tool 122 allows the developer to create a validation rule for a particular node of a structured data file, decide whether it is modal or not, and create the presentation and content of alerts.

Fig. 24 shows an exemplary custom validation screen 2400, including the development copy of the expense report electronic form 1702, the date data-entry field 1704, the properties box 1706 (obscured), the validation option box 1802, the OnBeforeChange event 1808, and the OnValidate event 1810. In this example, a developer can choose to input a script-based validation rule by selecting the OnBeforeChange event 1808 or the OnValidate event 1810. Event handlers are useful in associating script-based rules with nodes as discussed above and further discussed below.

Fig. 25 shows an exemplary script entry screen 2500. In this example, the system 100 continues to enable the developer to input a script-based validation rule by presenting the script entry screen 2500. To aid the developer in inputting script, the system 100 provides some of the script needed, which is shown in a script entry area 2502. By so doing, the system 100 makes it easier for the developer to input a script-based validation rule.

When a developer inputs script, the script can be written to include not only a validation rule, but also the information for alerts to a user and how those alerts are displayed.

At block 1622, once the developer has input the script, the system 100 receives the script. The system 100 saves the script, either alone or along with other files in the structured data file's solution.

This script described can, in one implementation involving XML data files, include the executable code identified by the handler-object attribute 1506 above.

Associating the Script-Based Validation Rule with Its Node

At block 1624, the system 100 associates the script with the appropriate node. The system can associate script-based validation rules (and the included alert information, if any) to a particular node through use of event handlers. The event handlers are stored in a file accessible by the real-time validation tool 122, such as in the solution. An event handler points to script that should be executed when data entered into a particular data-entry field is received, such as the event handler object having the handler-object name 1510 set forth in Fig. 15. The event handlers can point to the script through XPath expressions (such as the event-handler XPath expression 1514), or otherwise, so long as the event handler informs the real-time validation tool 122 of the correct script to execute for data received.

Using the event-handler XML element 1502 for example, the following declaration defines an event handler for a script-based validation rule that the real-time validation tool 122 will apply to a node associated with a "travelReport/Expenses" data-entry field in an electronic form.

```
<xsf:domEventHandler match="TravelReport/Expenses"
                     handlerObject="TravelExpenses" />
```

Event handlers can determine when the real-time validation tool 122 executes the script. One type of event handler is executed by the real-time

validation tool 122 before the real-time validation tool 122 allows the user of an electronic form to move on after entering data. The OnBeforeChange event handler 1808 is an example of this type of event handler. With this type of event handler, when data is entered and received by the real-time validation tool 122, but before the real-time validation tool 122 allows the user to edit another data-entry field, the real-time validation tool 122 (or the system 100) executes the script pointed to by the event handler.

Developers can use this type of event handler for errors that the developer wants to be modal. This type of event handler can be used for modal errors because the real-time validation tool 122 is able to determine, by executing the script, that the data entered is invalid before it has allowed the user to continue editing the electronic form. Thus, the developer can stop the user from continuing to edit the electronic form if the rule is violated.

Continuing the previous example, the following script is executed by the real-time validation tool 122 whenever any change (including deleting the node) is made to the "TravelReport/Expenses" data-entry field or its node (or any other node inside its hierarchy) but before the real-time validation tool 122 allows the user to continue editing the electronic form. This script-based validation rule is violated if the data received for the "TravelReport/Expenses" data-entry field is greater than 500, based on the script: "value of expense report != 500". If the data received is greater than 500, this script-based validation rule will cause the real-time validation tool 122 to return a modal error.

```
    function          TravelExpenses::onBeforeChange
(eventObj){
        if (eventObj.Source.Text != '500')
        {
            eventObj.ReturnMessage    =    'Correct
value is 500";
             return false;
         }
        return true;
    }
```

Another type of event handler is executed by the real-time validation tool 122 after the real-time validation tool 122 has allowed the user of an electronic form to move on after entering data. The OnValidate event handler 1810 is an example of this type of event handler. Developers can use this type of event handler for errors that the developer desires to be modeless. Errors that are programmatically added to the electronic form can constantly be checked by the real-time validation tool 122 and automatically cleared when the condition that triggered them no longer exists.

A Computer System

Fig. 26 shows an exemplary computer system that can be used to implement the processes described herein. Computer 2642 includes one or more processors or processing units 2644, a system memory 2646, and a bus 2648 that couples various system components including the system memory 2646 to processors 2644. The bus 2648 represents one or more of any of several types of

bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. The system memory 2646 includes read only memory (ROM) 2650 and random access memory (RAM) 2652. A basic input/output system (BIOS) 2654, containing the basic routines that help to transfer information between elements within computer 2642, such as during start-up, is stored in ROM 2650.

Computer 2642 further includes a hard disk drive 2656 for reading from and writing to a hard disk (not shown), a magnetic disk drive 2658 for reading from and writing to a removable magnetic disk 2660, and an optical disk drive 2662 for reading from or writing to a removable optical disk 2664 such as a CD ROM or other optical media. The hard disk drive 2656, magnetic disk drive 2658, and optical disk drive 2662 are connected to the bus 2648 by an SCSI interface 2666 or some other appropriate interface. The drives and their associated computer-readable media provide nonvolatile storage of computer-readable instructions, data structures, program modules and other data for computer 2642. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 2660 and a removable optical disk 2664, it should be appreciated by those skilled in the art that other types of computer-readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, random access memories (RAMs), read only memories (ROMs), and the like, may also be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk 2656, magnetic disk 2660, optical disk 2664, ROM 2650, or RAM 2652, including an

operating system 2670, one or more application programs 2672 (such as a real-time validation tool), other program modules 2674, and program data 2676. A user may enter commands and information into computer 2642 through input devices such as a keyboard 2678 and a pointing device 2680. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are connected to the processing unit 2644 through an interface 2682 that is coupled to the bus 2648. A monitor 2684 or other type of display device is also connected to the bus 2648 via an interface, such as a video adapter 2686. In addition to the monitor, personal computers typically include other peripheral output devices (not shown) such as speakers and printers.

Computer 2642 commonly operates in a networked environment using logical connections to one or more remote computers, such as a remote computer 2688. The remote computer 2688 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to computer 2642. The logical connections depicted in Fig. 26 include a local area network (LAN) 2690 and a wide area network (WAN) 2692. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When used in a LAN networking environment, computer 2642 is connected to the local network through a network interface or adapter 2694. When used in a WAN networking environment, computer 2642 typically includes a modem 2696 or other means for establishing communications over the wide area network 2692, such as the Internet. The modem 2696, which may be internal or external, is

connected to the bus 2648 via a serial port interface 2668. In a networked environment, program modules depicted relative to the personal computer 2642, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Generally, the data processors of computer 2642 are programmed by means of instructions stored at different times in the various computer-readable storage media of the computer. Programs and operating systems are typically distributed, for example, on floppy disks or CD-ROMs. From there, they are installed or loaded into the secondary memory of a computer. At execution, they are loaded at least partially into the computer's primary electronic memory. The invention described herein includes these and other various types of computer-readable storage media when such media contain instructions or programs for implementing the blocks described below in conjunction with a microprocessor or other data processor. The invention also includes the computer itself when programmed according to the methods and techniques described herein.

For purposes of illustration, programs and other executable program components such as the operating system are illustrated herein as discrete blocks, although it is recognized that such programs and components reside at various times in different storage components of the computer, and are executed by the data processor(s) of the computer.

Conclusion

The above-described XML elements and their accompanying processes enable validation of nodes of an XML data file. These XML elements are used to

specify validation rules, error messages, and executable code that can be used as part of validating nodes of an XML data file. Although the invention has been described in language specific to structural features and/or methodological acts, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the claimed invention.